

(19) 日本国特許庁 (J P)

(12) 公開特許公報 (A)

(11) 特許出願公開番号

特開平4-256192

(43) 公開日 平成4年(1992)9月10日

| (51) Int.Cl. ⁵ | 識別記号 | 庁内整理番号 | F I | 技術表示箇所 |
|---------------------------|---------|---------|-----|--------|
| G 0 6 K 9/36 | | 9073-5L | | |
| G 0 6 F 15/66 | 3 3 0 A | 8420-5L | | |
| G 0 6 K 9/00 | Z | 7737-5L | | |
| H 0 3 M 7/30 | | 8836-5J | | |

審査請求 未請求 請求項の数 2 (全 8 頁)

(21) 出願番号 特願平3-17664

(22) 出願日 平成3年(1991)2月8日

(71) 出願人 000005223

富士通株式会社

神奈川県川崎市中原区上小田中1015番地

(72) 発明者 千葉 広隆

神奈川県川崎市中原区上小田中1015番地

富士通株式会社内

(72) 発明者 岡田 佳之

神奈川県川崎市中原区上小田中1015番地

富士通株式会社内

(72) 発明者 吉田 茂

神奈川県川崎市中原区上小田中1015番地

富士通株式会社内

(74) 代理人 弁理士 大菅 義之 (外 1 名)

最終頁に続く

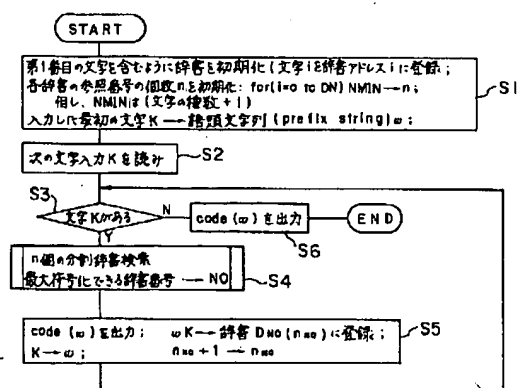
(54) 【発明の名称】 データ圧縮復元方式

(57) 【要約】

【目的】 本発明はデータの符号化によってデータを圧縮するデータ圧縮方式やその符号化によって得られた圧縮データを復元するデータ復元方式に関し、データの符号化におけるデータの圧縮率を高めることを目的とする。

【構成】 本発明は、入力文字に対して入力文字の繰り返しを表わせる最長の文字列を辞書から参照し、最長の文字列を表わせる辞書を選択し、その参照番号で符号化し (S4)、またその逆で復号化する。

本発明による符号化方式の符号化アルゴリズムのフローチャート



【特許請求の範囲】

【請求項1】 符号化済データを相異なる部分列に分けて辞書に登録しておき、入力データを前記辞書中の部分列のうちの最大長と一致するものの番号で指定して符号化する方式において、複数個設けられた辞書に対して検索を行い、最長の文字列を表わせる辞書を選択すると共に、該辞書の参照番号で指定して符号化することを特徴とするデータ圧縮方式。

【請求項2】 符号化済データを相異なる部分列に分けて辞書に登録しておき、入力データを前記辞書中の部分列のうちの最大長と一致するものの番号で指定して符号化したデータを復元する方式において、複数個設けた辞書に対して検索を行い、最長の文字列を復号できる辞書を選択し、該選択された辞書の参照番号から復号化を行うことを特徴とするデータ復元方式。

【発明の詳細な説明】

【0001】

【産業上の利用分野】 本発明はデータの符号化方式や復号化方式に係り、さらに詳しくは符号化によってデータを圧縮するデータ圧縮方式に係る。また、このデータ圧縮方式によって圧縮されたデータを復元するデータ復元方式に関する。

【0002】

【従来の技術】 近年、OA化の発展、更にはCPUの処理技術の向上により、カラーや白黒階調画像情報等のデータベースを計算機で扱うことが増えてきている。これらの画像情報のデータ量は1枚(1画面)あたり数Mバイトになり非常に大きいものである。このため蓄積や伝送等における画像情報を効率良く扱うため、データ圧縮を行って記憶すべきデータ量を減らしている。

【0003】 データ圧縮には様々な方式があるが、その一方式としてユニバーサル符号化がある。なお、本発明は文字コードの圧縮に限らず様々なデータに適用できるが、以下では説明のため情報理論の分野で呼称されているデータの1ワード単位を文字、データが複数ワードつながったものを文字列と呼ぶ。

【0004】 前述のユニバーサル符号化の代表的な方法として、Ziv-Lempel符号がある(詳しくは、例えば、宗像『Ziv-Lempelのデータ圧縮法』、情報処理、Vol. 26, No. 1, 1985年を参照のこと)。このZiv-Lempel符号では①ユニバーサル型と、②増分分解型(Incremental parsing)の2つのアルゴリズムが提案されている。

【0005】 ユニバーサル型のアルゴリズムは、演算量が多いが、高圧縮率が得られるという特徴を有している。この方式は、符号化データを過去のデータ系列の任意の位置から一致する最大長の系列に区切り(部分列)、過去の系列の複製として符号化する方法である。図6に示す如くPバッファとQバッファとが設けられ、Pバッファに符号化済みの入力データを格納し、Qバッファにこれから符号化するデータを格納する。そして、

Qバッファの系列はPバッファの系列をサーチし、Pバッファ中で一致する最大長の部分列を求める。そして、Pバッファ中でこの最大部分列を指定するための情報の組を符号化する。

【0006】 更に、ユニバーサル型アルゴリズムの改良としてLZSS符号がある。(T.C.Bell, "Better OPM/L Text Compression", IEEE Trans. on Commun., Vol. COM-34, No. 12, Dec.1986参照)。このLZSS符号では図5Aに示す如くPバッファ中の最大一致系列の開始位置を求め、一致する長さの組と、次のシンボルとをフラグで区別して符号量の少ない方で符号化するものである。

【0007】 一方、増分分解型アルゴリズムは、圧縮率ではユニバーサル型より劣るが、シンプルで計算も容易であるという特徴を有している。増分分解型Ziv-Lempel符号では、入力シンボルの系列を $x = a \ a \ b \ a \ b \ a \ b \ a \ a \cdots$ とすると、成分系列 $x = X_0 \ X_1 \ X_2 \cdots$ への増分分解は次のようにしている。まず X_1 を既成分の右端のシンボルを取り除いた最長の列とし、 $X = a \cdot a \ b \cdot a \ b \ a \cdot b \cdot a \ a \cdots$ としている。従って、 $X_0 = \lambda$ (空列)、 $X_1 = X_0 \ a$ 、 $X_2 = X_1 \ b$ 、 $X_3 = X_2 \ a$ 、 $X_4 = X_0 \ b$ 、 $X_5 = X_1 \ a$ 、 \cdots と分解できる。

【0008】 増分分解した各成分系列は既成分系列を用いて、図5Bに示すごとく各成分の成分のインデックスと次のシンボルを用いて符号化している。すなわち増分分解型アルゴリズムは、符号化パターンについて、過去に分解した部分列の内最大長を意識するものを求め、過去に分解した部分列の複製として符号化するものである。

【0009】 さらに、前述の増分分解型アルゴリズムの改良として、LZW符号がある。(T.A. Welch, "A Technique for High-Performance Data Compression", Computer, June 1984参照)。このLZW符号では、次のシンボルを次の部分列に組み込むようにしてインデックスのみで符号化できるようにしている。

【0010】 図7は従来のLZW符号化による処理フロー図である。LZW符号化処理においては、書き換え可能な辞書を有し、入力文字列を相異なる文字列(部分列)に分け、この文字列を出現した順に参照番号を付けて辞書に登録するとともに、現在入力している文字列を辞書に登録してある最長一致文字列の参照番号で表わして符号化するものである。

【0011】 先ず、処理S1で予め辞書に全文字につき1文字からなる文字列を初期値として登録してから後述する符号化を始める。また、処理S1では入力した最初の文字Kにより辞書を検索して最小番号 w を求め、これを語頭文字列とする。続いて処理S2で入力データの次の文字Kを読み込み、処理S3で全ての文字入力が終了したか否かをチェックする。入力文字が存在する、すな

わち文字Kが存在する時(Y)には語頭文字列 ω に処理S2で読み込んだ文字Kを加えた文字列(ωK)が辞書に存在するかどうかを求める。

【0012】判別処理S4で文字列(ωK)が辞書に存在しなければ(N)、処理S6によって処理1で求めた文字Kの参照番号 ω を符号語code(ω)として出力し、また文字列(ωK)を新たな参照番号として辞書に登録し、更に処理S2の入力文字Kを参照番号 ω に置き換えると共に辞書アドレスnをインクリメントして再度処理S2より実行する。

【0013】一方、処理S4で文字列(ωK)が辞書に存在する時(Y)には、文字列(ωK)を参照番号 ω に置き換え(S5)、再び処理S2に戻って処理S4において文字列 ωK が辞書から探せなくなるまで最大一致長の検索を続ける。

【0014】また、文字Kが判別処理S3において存在しないと判別した時(N)には処理S7によってcode(ω)を出力し終了(END)する。前述した処理を図9、10を参照して具体的に説明する。

【0015】図9の入力データINPUT SYMBOLSは左から右へと順次読む。最初の文字aを入力した時、辞書にはaの他に一致する文字列がないので、OUTPUT CODE 1(参照番号 ω)を符号語として出力する。そして、拡張した文字列abに参照番号4を付けて辞書に登録する。実際の辞書登録は図10の右側(ALTERNATE TABLE)に示すように文字列1bとして登録する。続いて2番目の文字bが文字列の先頭になる。辞書にはbの他に一致する文字がないので参照番号2を符号文字として出力し、同時に拡張した文字列baも辞書にないので文字列baを2aで表わし、参照番号5を付けて辞書に登録する。そして3番目のaが次の文字列の先頭になる。以下同様にこの処理を続ける。

【0016】図8は図7の復号化処理によって求めた圧縮データの復号化処理のフローチャートである。図8のLZW復号化処理においては、符号化と同様に予め辞書に全文字につき1文字からなる文字列を初期値として登録してから復号を始める。

【0017】先ず処理S11で最初の符号(参照番号)を読み込み、現在のCODEをOLDcodeとし、最初の符号は既に辞書に登録された1文字の参照番号いずれかに該当することから、入力符号CODEに一致する文字code(K)を探し出し、文字Kを出力する。なお、出力した文字Kは後の例外処理のためFINcharにセットする。

【0018】次に処理S12において、次の符号を読み込んでCODEをINcodeとしてセットする。続いて新たな符号があるかどうかを判別(S13)し、新たな符号がない時(N)には終了(END)する。また、存在する時(Y)には処理S13で入力した符号CODEが辞書に定義されているかどうかをチェックする(S14)。

通常入力した符号語は前回までの処理で辞書に登録されているため、続いて符号CODEに対する文字列code(ωK)を辞書から読み出し、文字Kを一時的にスタック(S16)し、参照番号code(ω)を新たな符号CODEとして再度処理S15より実行する。この処理S15、S16の手順を再帰的に参照番号 ω が1文字Kに至るまで繰り返し最後に処理S17において処理S16でスタックした文字をLIFO (Last In First Out) 形式でポップアップして出力する。また同時に前回使用した符号 ω と今回復元した文字列の最初の1文字Kを組み合わせた文字列に、新たな参照番号として辞書に登録する。

【0019】図11を参照してLZW復号化処理を具体的に説明する。最初の入力符号(INPUT CODE)は1であり、1文字a, b, cについては既に参照番号1, 2, 3として図11に示す如く辞書に登録されている。よって辞書の参照により符号1に一致する参照番号の文字列aに置き換えて出力する。次の符号2についても同様にし、文字bに置き換えて出力する。この時前回処理した符号1と今回復元した最初の1文字bとを組み合わせた文字列(1b)に新たな参照番号4を付加して辞書に登録する。

【0020】3番目の符号4は辞書の検索により求めた文字列1bから文字列abと置き換えて文字列abを出力する。同時に前回処理した符号2と今回復元した文字列の1番目の文字aとの組み合わせた文字列2a(=ba)に新たな参照番号5を付加して辞書に登録する。そして以下同様に繰り返すことにより復号がなされる。

【0021】図11のLZW復号化においては次の例外処理がある。この例外処理は例えば第6番目の入力符号8の復号にて生ずる。復号8は復号時に辞書に定義されておらず復号できない。この場合には前回処理した符号5に前回復元した文字列baの最初の1文字bを加えた文字列5bを求め、更に2ab=babと置き換えて出力する例外処理を行う。そして、文字列の出力後に前回の符号5に今回復元した文字列の1番目の文字bを加えた文字列5bに参照番号8を付加して辞書に登録する。

【0022】この例外処理は図7の復号化処理フローの処理S4、S8によって行われ、最終的に処理S7で文字列の出力と新たな文字列に参照番号を付加した辞書への登録が行われる。

【0023】尚、図8、図11のLZW復号化においては、復号側で符号を解釈しながら辞書をリアルタイムで作り出す場合を説明しているが、符号化の際に作られた辞書をそのまま復号化側にコピーして使用することで、復号化している場合もあり、この場合には復号側での例外処理は不要になる。

【0024】

【発明が解決しようとする課題】前述した従来のLZW符号においては、入力文字コード・データを相異なる文

字列に分けて符号化するとき、現在符号化中の各文字列は以前の文字列とは独立に出現するとして符号化する形式を用いている。しかしながら、従来の方式においては、辞書が1つであるため、その再現する文字列を表わす符号には冗長性を有し、圧縮率の低下を招いてた。しかしながら、データの増加に伴い、さらに圧縮率の高い方式が要求されている。

【0025】本発明は圧縮率を高めたデータ圧縮方式とその圧縮してデータを復元する復元方式を提供することを目的とする。

【0026】

【課題を解決するための手段とその作用】本発明は第1には符号化済データを相異なる部分列に分けて辞書に登録しておき、入力データを前記辞書中の部分列のうちの最大長と一致するものの番号で指定して符号化する方式におけるものである。

【0027】複数個設けた辞書に対して検索を行い、最長の文字列を表わせる辞書を選択して、選択された辞書の参照番号で指定して符号化する。入力文字に対して入力文字の繰り返しを表わせる最長の文字列を辞書から参照し、最長の文字列を表わせる辞書を選択し、その参照番号で符号化する。復号時には選択した辞書がわかるので辞書の選択指示を必要とせず、高圧縮率をなすことができる。

【0028】また、復号においては、複数個設けた辞書に対して検索を行い、最長の文字列を復号できる辞書を選択しその選択した辞書の参照番号から復号化を行う。この時、上述の最長の文字列を復号できる辞書を選択するので上述したデータ圧縮時の辞書の指示を必要とせず、圧縮における高効率化とそれによって圧縮したデータを復元することができる。

【0029】

【実施例】以下、図面を用いて本発明を詳細に説明する。本発明はデータ圧縮並びに復元方式におけるものであり、回路によって構成することも又プロセッサのソフトウェアによって処理を行うこともできる。以下ではプロセッサを用いた時のフローを用いて詳細に説明する。図1は本発明による符号化方式の符号化アルゴリズムのフローチャートである。本発明は複数(DN個)の各辞書D_i (i=1, ..., DN)に1文字からなる文字列全種を初期値として予め登録する。そして各辞書の参照番号の総数をn(i)で管理し、初期化のとき、DN個のn₁ (文字種+1)をセットする。1文字を入力した時はLZW符号と手順は同様である。また従来のLZW符号では辞書は1個だけであったのに対して本発明の実施例においては複数個の辞書に対して文字列の照合を行い、一番長い文字列を符号化できた辞書の参照番号を実際の符号化に使用している。使用された辞書D_{n₀}への登録後は、D_{n₀}の参照番号数を管理するn_{n₀}が1つインクリメントされる。この時文字列照合の結果文字列の長さ

が同じ長さになった場合には、乱数により辞書を決定する。

【0030】さらにその処理を詳細に説明する。入力バッファに圧縮すべきデータが入力すると、図1の処理を実行する。まず第1番目の文字を含むように辞書を初期化する。この初期化では文字1を辞書アドレス1に登録し、各辞書の参照番号の個数nを初期化する。これは
for (i=1 to DN) NM IN → n;
を表わすものである。但し、NM INは(文字の処理+1)である。

【0031】さらに入力した最初の文字Kを語頭文字列(prefix string)ω;として登録する(S1)。初期設定(S1)の後、次の文字入力Kを読み(S2)、つづいて文字Kがあるか否かを判別(S3)する。文字Kが存在する時(Y)にはn個分の分割辞書検索を行い、最大符号化できる辞書番号を求める。この検索は後述詳細に説明するが、辞書を指定しなくても最大符号化できる辞書番号は辞書に依存するので復号時にその辞書番号を指示しなくても求めることが出来る。ここで求めた辞書番号からその辞書に対しcode(ω)を出力し、ωKを辞書D_{n₀} (n_{n₀})に登録し、Kをωとしn_{n₀}+1をn_{n₀}とする。この処理(S5)の後再度判別(S3)より繰り返す。

【0032】判別(S3)において文字Kが存在しない時には処理(S5)によって今まで求めたコードを出力(S6)し、終了(END)する。前述した処理(S4)を更に詳細に表わすと図2の如くなる。処理(S4)を実行開始するとまず入力文字ポインタをテンポラリポインタレジスタtmpに格納する(S7)。続いてカウンタ(COUNT)に1を、また入力文字ポインタにテンポラリレジスタtmpの内容を格納する。処理(S7)における入力文字ポインタをテンポラリポインタレジスタtmpに格納するのはDN回繰り返す毎に入力文字ポインタの先頭を設定するのに必要とするその値を記憶するためである。すなわち、テンポラリポインタレジスタtmpの内容を入力文字ポインタに再度格納するのは順次繰り返される辞書に対応して入力文字ポインタを以後行うべき位置、すなわちテンポラリポインタレジスタtmpに格納されている値(処理を開始時の入力文字ポインタ)にするためである。

【0033】処理(S8)に続いてωKが辞書に存在するか否かを判別する(S9)。ωKが辞書に存在する時(Y)にはωKの値をωに格納し(S10)、つづいて次の文字Kを読むと共にカウント+1をカウントに格納する(S11)。そして再度処理(S9)より繰り返す。この繰り返しにより順次文字列が存在することとなる。一方、判別処理(S9)により存在する文字列がないと判別した時(N)にはそこで文字列が終了しているのでカウント(count)値を保存する(S12)。前述した処理(S8~S12)を順次繰り返し行う。この繰り返

返しは辞書の数DN分行う(S13)。そしてDN回行った後、保存したカウン트의最大値を選択(S14)し(同数の場合は乱数で選択)、辞書番号1をN0に格納しカウント+テンポラリの値を入力文字ポインタとする(S4)。そして、本処理を終了する。前述した処理によって各辞書対応で最大のカウン트가変化し、最大値に対応した辞書を選択するが、この選択は復号においても同様であり、図4に示す如くコードを出力する ω はその辞書に対応した ω であって、 ω やその一部が辞書を指示しているものではない。しかしながら、復号においてその指示が自動的になされるので辞書を指示する必要はなく、圧縮率を高めることができる。

【0034】図3は本発明における復号化のアルゴリズムのフローチャートである。復号化においても符号化と同様に例えばプロセッサによってその処理を行う。本発明の復号は符号化の逆の動作をするものであり、辞書の初期化は符号化と同様である。復号においては入力した符号CODEから参照番号 ω を復号した後、最長に復元される辞書から正式の辞書を決定して文字列を求め、符号化と同様に最長の文字列が多数求めた場合には乱数により決定する。この時、符号時と同じ乱数のシードを使用することにより符号側で符号化した辞書と同じ辞書を決定することができる。更にその処理を詳細に説明する。

【0035】圧縮したデータを復号化する際、まずバッファ等に圧縮すべきデータが格納される。この格納の後、図3におけるプログラムを実行する。まず処理を開始STARTすると、最初の符号を読み込み、 code^{-1} (CODE)をOLD ω とし、続いて $\omega = D(K)$ 、文字Kを出力、KをFINcharに格納する処理を行う(S21)。続いて次の入力コードを読み取る(S22)。そして新たな符号があるかを判別(S23)し、新たな符号がない(N)には終了(END)する。また、符号が存在する時(Y)には続いて code^{-1} (CODE)を ω とし、 ω をIN ω とする(S24)。続いてn個の分割辞書検索処理(S25)を実行し、最大符号化できる辞書番号を求める。そして辞書番号N0のSTACKが空になるまでSTACK TOPを出力し、POP STACKする。また、復元文字列の第1文字をFINcharに、また(OLD ω , K)を辞書D $_{n_0}$ (n_{s_0})に登録し、さらに $n_{s_0} + 1$ を n_{s_0} 、IN ω をOLD ω とする(S26)。そして再度(S22)より実行し順次これを繰り返す。

【0036】さらに前述したn個の分割辞書検索(S25)について図4で説明する。分割辞書検索処理(S25)を実行開始すると、先ずOLDcodeをTMP-OLDcodeに、FINcharをTMP-FINcharに、 ω をTMP- ω にそれぞれ格納する(S27)。この処理は複数の辞書を同一条件で検索する、すなわちそれぞれの辞書に対応して同一条件からスタートするようにするため

に一時的に記憶するものである(S27)。続いてN回繰り返すためのイニシャル処理を行う(S28)。この処理は前述した処理S27と逆の処理であり、各テンポラリに格納したTMP-OLDcodeのデータをOLDcodeに、TMP-FINcharのデータをFINcharに、TMP- ω を ω に、カウンタを1にする処理である(S29)。続いて判別処理(S29)で ω とnとを比較し $\omega = n$ であるならば、FINcharを出力、OLDcodeをCODEに、D(IN ω)を(OLDcode, FINchar)に格納する(S31)。また、 ω がnより小さいならば辞書D $_{n_0}$ より $\omega' = K = D_{n_0}(\omega)$ を判別し、等しい時(Y)にはKをPUSH STACKに、 ω' を ω に、CONT+1をCONTに設定し、再度判別(S30)を実行する。また、判別(S30)において等しくないときと判別した時(N)にはOLDcode, FINchar, ω , CONT, STACKの内容を保存する(S33)。また、判別(S29)において ω がカウンタ値nより大であるときには前述の処理を行わず次の辞書の処理に移る。この前述した処理(S28~S33)を分割辞書数分DN回繰り返す。そしてその繰り返しの後最大のカウンタ値を選択し同数の場合は乱数で選び辞書番号1をナンバーとして出力する。また保存してあるOLDcode, FINchar, ω を設定する(S34)。

【0037】なお、例外処理とし、符号化時に選択された最大一致長系列を示す参照番号が、他の分割辞書ですでに使用され、なおかつ、現在選択された系列よりも長い系列を表す場合には、符号化する参照番号の前に選択された辞書の番号を示す制御コードを符号化し、それに続き、参照番号を符号化する。復号化においても、選択番号を示す制御コードを検出したなら、指定された分割辞書により復元を行う。

【0038】以上のような動作により、復号時に符号値と同様の処理を逆に行い複数の辞書が存在しても、その辞書を指示されなくても復号側で求めることができ、例えば圧縮して転送する等の場合にその圧縮率が高まり転送効率を得ることが出来る。本発明の実施例ではプロセッサによる処理を用いているがこれに限らず、例えば回路等によって行うことも可能である。

【0039】

【発明の効果】以上述べたように本発明によれば、複数の分割した辞書により文字列を符号化できるため、従来のLZW符号より高い圧縮率が得られるとともに、符号が参照番号のみで表わされる簡単なアルゴリズムで実行できる。

【図面の簡単な説明】

【図1】本発明による符号化方式の符号化アルゴリズムのフローチャートである。

【図2】本発明による最大文字列検索のフローチャートである。

【図3】本発明による復号化方式の復号化アルゴリズム

のフローチャートである。

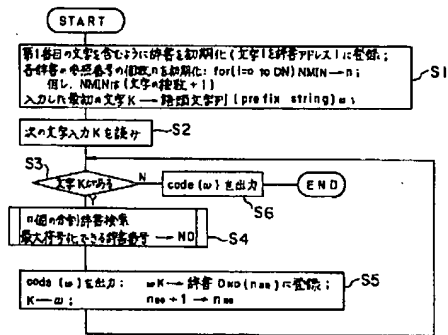
【図4】本発明による分割辞書検索のフローチャートである。

【図5】ユニバーサル符号化のアルゴリズムである。

【図6】ユニバーサル型ZL符号の符号化の原理図である。

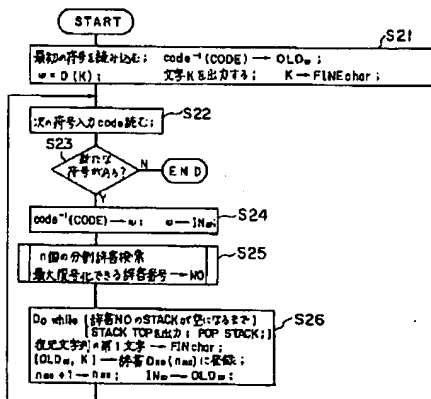
【図1】

本発明による符号化方式の符号化アルゴリズムのフローチャート



【図3】

本発明による符号化方式の復号化アルゴリズムのフローチャート



【図7】従来のLZW符号化処理フロー図である。

【図8】従来のLZW復号化処理フロー図である。

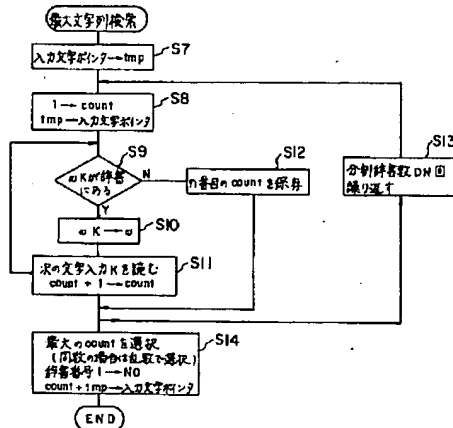
【図9】LZW符号化説明図である。

【図10】辞書構成例の説明図である。

【図11】LZW復号化説明図である。

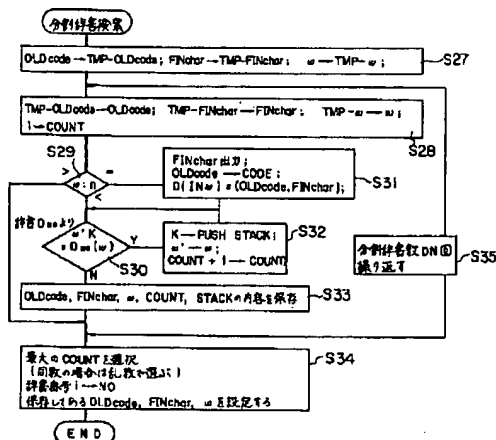
【図2】

本発明による符号化方式の符号化アルゴリズムのフローチャート



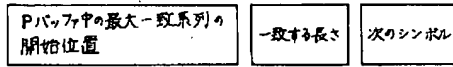
【図4】

分割辞書検索のフローチャート

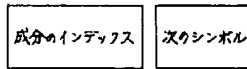


【図5】

ユニバーサル符号化のアルゴリズム



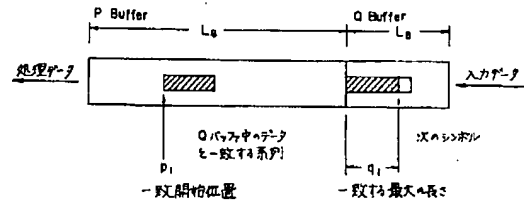
(A)



(B)

【図6】

ユニバーサル型ZL符号の符号化の原理図

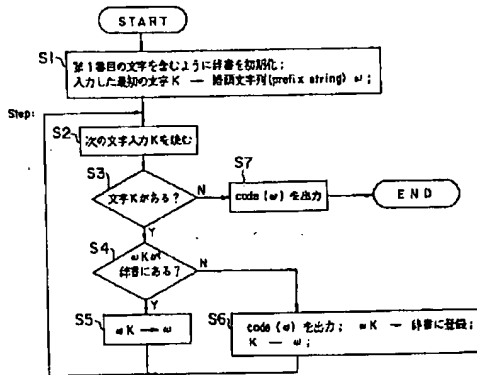


【図8】

従来のLZW符号化処理フロー図

【図7】

従来のLZW符号化処理フロー図



【図9】

LZW符号化説明図

| INPUT SYMBOLS | a | b | b | c | b | b | c | b | c | a | a | a | a | a | a | a | a |
|---------------------------|---|---|---|----|----|---|---|----|----|---|---|---|---|---|---|---|---|
| OUTPUT CODES | 1 | 2 | 4 | 3 | 5 | 8 | 1 | 10 | 11 | | | | | | | | |
| NEW STRING ADDED TO TABLE | 4 | 6 | 8 | 10 | 12 | | | | | | | | | | | | |

【図10】

辞書構成例の説明図

| STRING TABLE (説明用) | | ALTERNATE TABLE (実用版) | |
|-----------------------|----|--------------------------|----|
| a | 1 | o | 2 |
| b | 2 | c | 3 |
| c | 3 | | 4 |
| cb | 4 | bb | 5 |
| bc | 5 | cb | 6 |
| cb | 6 | cb | 7 |
| cb | 7 | cb | 8 |
| cb | 8 | cb | 9 |
| cb | 9 | cb | 10 |
| cb | 10 | cb | 11 |
| cb | 11 | cb | 12 |
| cb | 12 | cb | 13 |

(8)

特開平4-256192

【図11】

LZW符号化説明図

| | | | | | | | | | |
|-----------------------------|---|---|----|---|----|-----|----|----|-----|
| INPUT CODES | 1 | 2 | 4 | 3 | 5 | 8 | 1 | 10 | 11 |
| | V | V | V | V | V | V | V | V | V |
| | a | b | b | c | 2a | 5b | a | 1a | 10a |
| | | | V | | V | | | V | V |
| | | | a | | b | 2a | | a | 1a |
| OUTPUT DATA | a | b | ab | c | ba | bab | a | aa | aaa |
| STRING ADDED TO TABLE | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | |

フロントページの続き

(72)発明者 中野 泰彦
神奈川県川崎市中原区上小田中1015番地
富士通株式会社内

PAT-NO:

JP404256192A

DOCUMENT-IDENTIFIER: JP 04256192 A

TITLE: SYSTEM FOR COMPRESSING AND RESTORING DATA

PUBN-DATE: September 10, 1992

INVENTOR-INFORMATION:

NAME

COUNTRY

CHIBA, HIROTAKA

OKADA, YOSHIYUKI

YOSHIDA, SHIGERU

NAKANO, YASUHIKO

ASSIGNEE-INFORMATION:

NAME

COUNTRY

FUJITSU LTD

N/A

APPL-NO:

JP03017664

APPL-DATE:

February 8, 1991

INT-CL (IPC):

IPCS G06K009/36 , G06F015/66 , G06K009/00 , H03M007/30##IPCE##

US-CL-CURRENT: CLS\341/95\CLE

ABSTRACT:

PURPOSE: To enhance a data compression ratio by retrieving plural dictionaries, selecting the dictionary which expresses a longest character string, executing designation through the use of the reference number of the dictionary and executing encoding.

CONSTITUTION: The plural prepared dictionaries are retrieved, the dictionary which expresses the longest character string is selected, designation is executed by the reference number of the selected dictionary and encoding is executed.

The longest character string which expresses the repetition of an input character as against the input character is referred to the dictionary, the dictionary which expresses the longest character string is selected and the encoding is executed (S4) by the reference number. In the case of decoding, the plural dictionaries are retrieved, the dictionary which decodes the longest character string is selected and decoding is executed by the reference number of the selected dictionary.

COPYRIGHT: (C)1992,JPO&Japio